

GO-02 08: Основы языка. Обработка ошибок

Описание:

На данный момент вы уже знаете, что в Go для критических ситуаций реализован механизм паники и восстановления. Но как обстоят дела с обычными ошибками, которые являются неотъемлемой частью приложения?

В Go ошибки возвращаются функциями в явном виде, с передачей информации об ошибке. Мы уже упоминали о том, функции поддерживают возвращение нескольких значений. Одним из этих значений очень часто бывает ошибка, которая, как правило, возвращается последней.

```
package main

import "fmt"

func main() {
    _, err := checkNum(11)
    if err != nil {
        fmt.Println("invalid num:", err)
    }
}

func checkNum(x int) (valid bool, err error) {
    if x > 0 && x > 10 {
        return false, fmt.Errorf("out of positive range: %v", x)
    }
    if x < 0 && x < -10 {
        return false, fmt.Errorf("out of negative range: %v", x)
    }

    return true, nil
}
```

На примере выше в сигнатуре функции `checkNum()` указано, что она возвращает два значения. Первое значение булево, может принимать `false`, если входной параметр выходит за пределы какого-то интервала. В этом случае хорошо бы вернуть ошибку,

чтобы можно было отреагировать на нее. Поэтому вторым параметром функция возвращает ошибку, о чём мы явно указали в сигнатуре.

Тип `error` является встроенным интерфейсом. Если значение ошибки не инициализировано, то в этом случае нужно вернуть `nil`. Когда возникновение ошибки является ожидаемым, как в нашем примере, то ошибку можно создать при помощи функции стандартного пакета `fmt.Errorf()`, добавив необходимую информацию.

Также важно понимать, что возвращаемое значение ошибки может проходить через несколько функций в качестве значения и эти функции могут ее каким-то образом обрабатывать, добавляя сообщения и дополнительные данные контекста вызова. Именно поэтому хорошей практикой работы с ошибками является написание сообщений без использования заглавных букв и символов переноса строки. В качестве разделителя в основном используется двоеточие. В итоге, когда ошибка пройдет через все вызовы и вы будете просматривать логи работы приложения, запись об ошибке окажется максимально полной и информативной, что позволит однозначно понять причину ее возникновения.

Запустив наш пример, можно увидеть довольно емкое сообщение.

`invalid num: out of positive range: 11`

В Go принято обрабатывать ошибки явно и сразу после вызова с возможным возвратом ошибки. Простые конструкции управления потоком выполнения позволяют писать понятный код.

Полезные ссылки:

- [Error handling and Go](#)
- [Golang — изящная обработка ошибок](#)
- [Creating Custom Errors in Go](#)
- [Golang Error Handling — Best Practice in 2020](#)

Задание:

1. Ознакомьтесь со стандартными пакетами `log`, `errors`.
2. Создайте в проекте `module02` новую ветку `08_task`.
3. Создайте новую директорию и файл `main.go`.
4. Скачайте файл [in.txt](#) и скопируйте его в директорию `data` рядом с файлом `main.go`.
5. Напишите программу, которая построчно считывает файл.
6. Выведите в консоль количество строк в формате `Total strings: %d`.
7. Корректно обработайте в отложенном вызове ошибки закрытия файловых дескрипторов.
8. Корректно обработайте ошибку окончания файла `EOF`.
9. Зафиксируйте изменения в ветке и отправьте их в удаленный репозиторий проекта.

10. В качестве ответа пришлите ссылку на merge request в ветку master вашего проекта ветки 08_task.